The Extensible Markup Language (XML) Configuration Access Protocol (XCAP)
draft-ietf-simple-xcap-02

Status of this Memo

Copyright Notice

Abstract

This specification defines the Extensible Markup Language (XML) Configuration Access Protocol (XCAP). XCAP allows a client to read, write and modify application configuration data, stored in XML format on a server. XCAP is not a new protocol. XCAP maps XML document sub-trees and element attributes to HTTP URIs, so that these components can be directly accessed by HTTP.

Table of Contents

```
   1.       Introduction . . . . . . . . . . . . . . . . . . . . . . . .  4
   2.       Overview of Operation  . . . . . . . . . . . . . . . . . . .  5
   3.       Terminology  . . . . . . . . . . . . . . . . . . . . . . . .  6
   4.       Application Usages . . . . . . . . . . . . . . . . . . . . .  7
   4.1      Application Usage ID (AUID)  . . . . . . . . . . . . . . . .  7
   4.2      Data Validation  . . . . . . . . . . . . . . . . . . . . . .  7
   4.3      Data Semantics . . . . . . . . . . . . . . . . . . . . . . .  8
   4.4      Naming Conventions . . . . . . . . . . . . . . . . . . . . .  8
   4.5      Data Interdependencies . . . . . . . . . . . . . . . . . . .  8
   4.6      Authorization Policies . . . . . . . . . . . . . . . . . . .  8
   4.7      Data Extensibility . . . . . . . . . . . . . . . . . . . . .  9
   4.7.1    XML Schema . . . . . . . . . . . . . . . . . . . . . . . . . 10
   4.8      Documenting Application Usages . . . . . . . . . . . . . . . 10
   5.       URI Construction . . . . . . . . . . . . . . . . . . . . . . 11
   5.1      Identifying the XML Document . . . . . . . . . . . . . . . . 11
   5.2      Identifying the XML Nodes  . . . . . . . . . . . . . . . . . 12
   6.       Client Operations  . . . . . . . . . . . . . . . . . . . . . 15
   6.1      Create or Replace a Document . . . . . . . . . . . . . . . . 15
   6.2      Delete a Document  . . . . . . . . . . . . . . . . . . . . . 15
   6.3      Fetch a Document . . . . . . . . . . . . . . . . . . . . . . 15
   6.4      Create or Replace an Element . . . . . . . . . . . . . . . . 16
   6.5      Delete an Element  . . . . . . . . . . . . . . . . . . . . . 17
   6.6      Fetch an Element . . . . . . . . . . . . . . . . . . . . . . 17
   6.7      Create or Replace an Attribute . . . . . . . . . . . . . . . 17
   6.8      Delete an Attribute  . . . . . . . . . . . . . . . . . . . . 18
   6.9      Fetch an Attribute . . . . . . . . . . . . . . . . . . . . . 18
   6.10     Read/Modify/Write Transactions . . . . . . . . . . . . . . . 19
   6.11     Reading Server Assigned Data . . . . . . . . . . . . . . . . 19
   7.       Server Behavior  . . . . . . . . . . . . . . . . . . . . . . 21
   7.1      POST Handling  . . . . . . . . . . . . . . . . . . . . . . . 21
   7.2      PUT Handling . . . . . . . . . . . . . . . . . . . . . . . . 22
   7.2.1    Detailed Conflict Reports  . . . . . . . . . . . . . . . . . 23
   7.2.1.1  XML Schema . . . . . . . . . . . . . . . . . . . . . . . . . 25
   7.3      GET Handling . . . . . . . . . . . . . . . . . . . . . . . . 27
   7.4      DELETE Handling  . . . . . . . . . . . . . . . . . . . . . . 28
   7.5      Managing Etags . . . . . . . . . . . . . . . . . . . . . . . 28
   8.       Examples . . . . . . . . . . . . . . . . . . . . . . . . . . 30
   9.       Security Considerations  . . . . . . . . . . . . . . . . . . 33
   10.      IANA Considerations  . . . . . . . . . . . . . . . . . . . . 34
   10.1     XCAP Application Usage IDs . . . . . . . . . . . . . . . . . 34
   10.2     application/xml-fragment-body MIME Type  . . . . . . . . . . 34
   10.3     application/xml-attribute-value MIME Type  . . . . . . . . . 35
   10.4     application/xcap-error+xml MIME Type . . . . . . . . . . . . 36
   10.5     URN Sub-Namespace Registration for
            urn:ietf:params:xml:ns:xcap-must-understand  . . . . . . . . 37
   10.6     URN Sub-Namespace Registration for
```

urn:ietf:params:xml:ns:xcap-error 38
10.7 XCAP Error Schema Registration 38
10.8 XCAP Mandatory Namespace Schema Registration 39
11. Acknowledgements 40
Normative References 41
Informative References 43
Author's Address 44
Intellectual Property and Copyright Statements 45

1. Introduction

In many communications applications, such as Voice over IP, instant messaging, and presence, it is necessary for network servers to access per-user information in the process of servicing a request. This per-user information resides within the network, but is managed by the end user themselves. Its management can be done through a multiplicity of access points, including the web, a wireless handset, or a PC application.

Examples of per-user information are presence [17] authorization policy and presence lists. Presence lists are lists of users whose presence is desired by a watcher. One way to obtain presence information for the list of is to subscribe to a resource which represents that list [20]. In this case, the Resource List Server (RLS) requires access to this list in order to process a SIP [15]SUBSCRIBE [25] request for it. Another way to obtain presence for the users on the list is for a watcher to subscribe to each user individually. In that case, it is convenient to have a server store the list, and when the client boots, it fetches the list from the server. This would allow a user to access their resource lists from different clients.

Requirements for manipulation of presence lists and authorization policies have been specified by the SIMPLE working group [21].

This specification describes a protocol that can be used to manipulate this per-user data. It is called the Extensible Markup Language (XML) Configuration Access Protocol (XCAP). XCAP is not a new protocol. Rather, it is a set of conventions for mapping XML documents and document components into HTTP URIs, rules for how the modification of one resource affects another, data validation constraints, and authorization policies associated with access to those resources. Because of this structure, normal HTTP primitives can be used to manipulate the data. XCAP is based heavily on ideas borrowed from the Application Configuration Access Protocol (ACAP) [23], but it is not an extension of it, nor does it have any dependencies on it. Like ACAP, XCAP is meant to support the configuration needs for a multiplicity of applications, rather than just a single one.

2. Overview of Operation

Each application that makes use of XCAP specifies an application usage (Section 4). This application usage defines the XML schema [2] for the data used by the application, along with other key pieces of information. The principal task of XCAP is to allow clients to read, write, modify, create and delete pieces of that data. These operations are supported using HTTP 1.1 [5]. An XCAP server acts as a repository for collections of XML documents. There will be documents stored for each application. Within each application, there are documents stored for each user. Each user can have a multiplicity of documents for a particular application. To access some component of one of those documents, XCAP defines an algorithm for constructing a URI that can be used to reference that component. Components refer to any subtree of the document, or any attribute for any element within the document. Thus, the HTTP URIs used by XCAP point to pieces of information that are finer grained than the XML document itself.

With a standardized naming convention for mapping components of XML documents to HTTP URIs, the basic operations for accessing the data are provided by existing HTTP primitives. Reading one of the components is accomplished with HTTP GET, creating or modifying one of the components is done with an HTTP PUT, and removing one of the components is done with an HTTP DELETE. To provide atomic read/modify/write operations, HTTP entity tags are used.

3. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [6] and indicate requirement levels for compliant implementations.

4. Application Usages

A central concept in XCAP is that of an application usage. An application usage defines the way in which a specific application makes use of XCAP.

4.1 Application Usage ID (AUID)

Each application usage is associated with a name, called an AUID. This name uniquely identifies the application usage, and is different from all other AUIDs. AUIDs exist in one of two namespaces. The first namespace is the IETF namespace. This namespace contains a set of tokens, each of which is registered with IANA. These registrations occur with the publication of standards track RFCs [24] based on the guidelines in Section 10. The second namespace is the vendor-proprietary namespace. Each AUID in that namespace is prefixed with the reverse domain name name of the organization creating the AUID, followed by a period, followed by any vendor defined token. As an example, the example.com domain can create an AUID with the value "com.example.foo" but cannot create one with the value "org.example.foo". AUIDs within the vendor namespace do not need to be registered with IANA. The vendor namespace is also meant to be used in lab environments where no central registry is needed. The syntax for AUIDs, expressed in ABNF [11] (and using some of the BNF defined in RFC 2396 [12]) is:

```
AUID                =  global-auid / vendor-auid
global-auid         =  auid
auid                =  alphanum / mark
vendor-auid         =  rev-hostname "." auid
rev-hostname        =  toplabel *( "." domainlabel  )
domainlabel         =  alphanum
                       / alphanum *( alphanum / "-" ) alphanum
toplabel            =  ALPHA / ALPHA *( alphanum / "-" ) alphanum
```

4.2 Data Validation

One of the responsibilities of the server is to validate the data generated by the client. This is done using two mechanisms. Firstly, all application usages MUST describe their document contents using XML schema [2]. Unfortunately, XML schemas cannot represent every form of data constraint. As an example, one XML element may contain an integer which defines the maximum number of instances of another element. This constraint cannot be represented with XML schema. However, such constraints may be important to the application usage. The application usage defines any additional constraints beyond those

in the schema.

4.3 Data Semantics

For each application usage, the data present in the XML document has a well defined semantic. The application usage defines that semantic, so that a client can properly construct a document in order to achieve the desired result.

4.4 Naming Conventions

In addition to defining the meaning of the document in the context of a particular application, and application usage has to specify how elements in that application obtain the various documents necessary for operation of that application. In particular, what the relevant URIs are that point to documents used by the application.

As an example, one application that can make use of XCAP is a SIP event list subscription [20]. In this application, an entity is defined called a Resource List Server (RLS). When the RLS receives a subscription to a SIP URI that represents a list, it "expands" the list and subscribes to its members. The XCAP resource list application usage [22] specifies how the RLS uses XCAP to find the XML document that defines the contents of the list.

These conventions are defined as naming conventions.

4.5 Data Interdependencies

In many cases, when a user modifies an XCAP resource, other data managed by the server needs to change as well. Such interdependencies are application usage dependent. As an example, when a user performs a PUT operation to create a new presence list, the server may need to fill in the URI associated with that list. These interdependencies need to be specified by the application usage.

4.6 Authorization Policies

By default, an XCAP server will only allow a user to access (read, write, delete or modify) their own documents. The application usage can specify differing default authorization policies. An application usage can also specify whether another application usage is used to define the authorization policies. An application usage for setting authorization policies can also be defined subsequent to the definition of the the main application usage. In such a case, the main application usage needs only to specify that such a usage will be defined in the future.

4.7 Data Extensibility

An XCAP server MUST understand an application usage in order to process an HTTP request made against a resource for that particular application usage. However, it is not required for the server to understand all of the contents of a document used by an application usage. A server is required to understand the baseline schema defined by the application usage. However, those schemas can define points of extensibility where new content can be added from other namespaces and corresponding schemas. Sometimes, the server will understand those namespaces and therefore have access to their schemas. Sometimes, it will not.

A server MUST allow for documents that contain elements from namespaces not known to the server. In such a case, the server cannot validate that such content is schema compliant; it will only verify that the XML is well-formed.

Unfortunately, it may be the case that a client needs the server to understand these new namespaces in order to process a document. This will be the case when the new content contains data interdependcies that the server has to understand. To allow for this, this specification defines an XML element called "mandatory-ns". A server will look for the presence of this element as the child of the root node of any document. If it finds it, the server will make sure that it is familiar with any namespaces (and their corresponding schemas) listed there.

The implication is that the schema for all XCAP application usages MUST allow for the "mandatory-ns" element to be present as a child of the root node of any document. This can be done by explicitly importing its namespace and including it in the schema, or allowing elements from other namespaces to be present in the schema as children of the root node.

4.7.1 XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="urn:ietf:params:xml:ns:xcap-must-understand"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:tns="urn:ietf:params:xml:ns:xcap-must-understand"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="mandatory-ns">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="ns" type="xs:anyURI" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

4.8 Documenting Application Usages

Application usages are documented in specifications which convey the information described above. In particular, an application usage specification MUST provide the following information:

Application Usage ID (AUID): The application usage MUST register the AUID using the IANA procedures defined in Section 10.

MIME Type: Each application usage will have a MIME type for its documents. This can either be an existing MIME type, or a new one registered by the application usage.

XML Schema: The schema for documents used by the application.

Additional Constraints: Any constraints that can not be represented by the XML schema.

Data Semantics:

Naming Conventions:

Resource Interdependencies:

Authorization Policies: If the application usage changes the default authorization policies, it should specify that. If not, it should specify that the default is used.

5. URI Construction

In order to manipulate a piece of configuration data, the data must be represented by an HTTP URI. XCAP defines a specific naming convention for constructing these URIs. In particular, the host part identifies the XCAP server. The abs_path component of the HTTP URI identifies the specific piece of data to be modified. This path is broken into a two parts. The first part identifies the particular XML document. XCAP servers organize XML documents in a specific hierarchical fashion, as described in Section 5.1. The second part of the path is called a node selector. When present, it contains an XML component identifier formatted according to Section 5.2. The node selector identifies the specific component of the XML document. The HTTP URI without the node selector is called the document URI.

Note that there is nothing in the grammar for the HTTP URI that separates the document URI from the node selector. The path extends naturally from the document into the XML hierarchy within the document. Separating the two components is something a server can do based on its awareness of the structure of the document directory.

5.1 Identifying the XML Document

XCAP mandates that a server MUST organize documents according to a defined hierarchy. The root of this hierarchy is an HTTP URI called the XCAP services root URI. This URI identifies the root of the tree within the domain where all XCAP documents are stored. It can be any valid HTTP URL, but MUST NOT contain a query string. As an example, http://xcap.example.com/services might be used as the XCAP services root URI within the example.com domain. Typically, the XCAP services root URI is provisioned into client devices for bootstrapping purposes.

Beneath the XCAP services root URI is a tree structure for organizing documents. The first level of this tree consists of the XCAP AUID. So, continuing the example above, all of the documents used by the presence list application would be under http://xcap.example.com/services/presence-lists.

It is assumed that each application will have data that is set by users, and/or it will have global data that applies to all users. As a result, within the directory structure for each application usage, there are two sub-trees. One, called "users", holds the documents that are applicable to specific users, and the other, called "global", holds documents applicable to all users.

Within the "users" tree are zero or more sub-trees, each of which identifies documents that apply to a specific user. XCAP does not

itself define what it means for documents to "apply" to a user, beyond specification of a baseline authorization policy, described below in Section 7. Each application usage can specify additional authorization policies which depend on data used by the application itself.

The remainder of the URI (the path following "global" or the specific user) is not constrained by this specification. The application usage MAY introduce constraints, or may allow any structure to be used.

5.2 Identifying the XML Nodes

The node selector specifies specific nodes of the XML document which are to be accessed. A node refers to either an XML element or an attribute of an element. The node selector is an expression which identifies an element or attribute. Its grammar is:

```
node-selector          =  element-selector ["/" attribute-selector]
element-selector       = step *( "/" step)
step                   = by-name / by-pos / by-attr
by-name                = QName      ; from XML Namespaces
by-pos                 = QName "[" position "]"
position               = 1*DIGIT
by-attr                = QName "[" "@" att-name "=" <">
                           att-value <"> "]"
att-name               = QName
att-value              = AttValue   ; from XML specification
attribute-selector     = "@" att-name
```

The QName grammar is defined the XML namespaces specification [3].

The node selector is based on the concepts in XPath [9]. Indeed, the node selector expression happens to be a valid XPath expression. However, XPath provides a set of functionality far richer than is needed here, and its breadth would introduce complexity into XCAP.

To determine the XML element or attribute selected by the node selector, processing begins at the root of the XML document. The first step in the element selector is then taken. Each step chooses a specific XML element within the current document context. The document context is the point within the XML document from which a specific step is evaluated. The document context begins at the root of the document. When a step determines an element within that context, that element becomes the new context for evaluation of the next step. Each step can select an element by its name, by a combination of name and attribute value, or by name and position. If the step is attempting selection by name, the server looks for all

elements within the current context with that name. Name matching is performed as described below. If there is more than one element with the specified name, the result is considered a no-match.

If the step is attempting selection by name and attribute, the server looks for all elements within the current document context with that name. Of those that match, it looks for ones that have the given attribute name, where that attribute has the given value. If there is no match, or if more than one element matches, the result is considered a no-match. Note that elements cannot be selected based on any namespace attributes. Any such attributes are effectively ignored in terms of the matching operations defined here.

If the step is attempting selection by name and position, the server looks for all elements within the current document context with that name. These are then sorted in document order, as defined by Xpath. The position-th element is then selected. If there are fewer than position number of elements with that name, the result is considered a no-match.

Once the last step is executed, if there is no attribute selector, the result of the node selection is the last selected element. If there is an attribute selector, the server checks to see if there is an attribute with that name within the currently selectoed element. If there is not, the result is considered a no-match. Otherwise, that attribute is selected. Note that namespace attributes cannot be selected.

Matching of element names and attributes is performed by expanding them into the expanded name form, as described in XML Namespaces, and then performing the comparison of the results. When evaluating the QNames in the node selector, the default namespace and namespace definitions from the document URI apply.

Comments, text content, and processing declarations in the XML document cannot be selected by the expressions defined here. Of course, if such information is present in a document, and a user selects an XML element enclosing that data, that information would be included in a resulting GET, for example.

As an example, consider the following XML document:

```
<?xml version="1.0"?>
   <watcherinfo xmlns="urn:ietf:params:xml:ns:watcherinfo"
                version="0" state="full">
     <watcher-list resource="sip:professor@example.net" package="presence">
       <watcher status="active"
                id="8ajksjda7s"
                duration-subscribed="509"
                event="approved" >sip:userA@example.net</watcher>
       <watcher status="pending"
                id="hh8juja87s997-ass7"
                display-name="Mr. Subscriber"
                event="subscribe">sip:userB@example.org</watcher>
     </watcher-list>
   </watcherinfo>
```

The node selector "watcherinfo/watcher-list/ watcher[@id="8ajksjda7s"]" would select the following XML element:

```
       <watcher status="active"
                id="8ajksjda7s"
                duration-subscribed="509"
                event="approved" >sip:userA@example.net</watcher>
```

6. Client Operations

An XCAP client is an HTTP 1.1 compliant client. Specific data manipulation tasks are accomplished by invoking the right set of HTTP methods with the right set of headers on the server. This section describes those in detail

6.1 Create or Replace a Document

To create or replace a document, the client constructs a URI that references the location where the document is to be placed. This URI MUST NOT contain a NodeSelector component. The client then invokes a PUT method on that URI.

The content in the request MUST be an XML document compliant to the schema associated with the application usage defined by the URI. For example, if the client performs a PUT operation to http://xcap.example.com/services/presence-lists/users/joe/mybuddies, presence-lists is the application unique ID, and the schema defined by it would dictate the body of the request. The MIME content type SHOULD be as specific as possible. For example, "application/resource-lists+xml" for a resource list [22], instead of just "application/xml".

If the Request-URI identifies a document that already exists in the server, the PUT operation replaces that document with the content of the request. If the Request-URI does not identify an existing document, the document is created on the server at that specific URI.

If the result of the PUT is a 200 or 202 response, the operation was successful. If it was a 409, the user performed some action which resulted in an invalid document. The 409 response may contain an XML body, formatted according to the schema in Section 7.2.1.1, which provides further information on the nature of the error. The client MAY use this information to try and alter the request so that this time, it might succeed. The client SHOULD NOT simply retry the request without changing some aspect of it.

6.2 Delete a Document

To delete a document, the client constructs a URI that references the document to be deleted. By definition this URI will not contain a NodeSelector component. The client then invokes a DELETE operation on the URI to delete the document.

6.3 Fetch a Document

As one would expect, fetching a document is trivially accomplished by

performing an HTTP GET request with the Request URI set to the document to be fetched. When a client fetches a document, and there is an older version cached, it is useful for clients to perform conditional GETs using the If-Match header field, in order to reduce network usage if the cached copy is still valid. An HTTP server MUST return Etags for entities that represent resources managed by XCAP.

6.4 Create or Replace an Element

To create or replace an XML element within an existing document, the client constructs a URI whose document URI points to the document to be modified. The node selector MUST be present in the URI. The node selector is constructed such that, if the element was added to the document as desired by the client, the node selector would select that element.

The client then invokes the HTTP PUT method. The content in the request MUST be an XML element. Specifically, it contains the element, starting with the opening bracket for the begin tag for that element, including the attributes and content of that element (whether it be text or other child elements), and ending with the closing bracket for the end tag for that element. The MIME type in the request SHOULD be "application/xml-fragment-body", defined in Section 10.2. The server will insert the element (including all its attributes and its content) into the document such that the node selector, if evaluated by the server, would return the content present in the request. If the node selector, when evaluated against the current document, results in a no-match, the server performs a creation operation. If the node selector, when evaluated against the current document, is a match for an element in the current document, the server replaces it with the content of the PUT request. This replacement is complete; that is, the old element (including its attributes and content) are removed, and the new one, including its attributes and content, is put in its place. The client SHOULD be certain, before making the request, that the resulting modified document will also be conformant to the schema.

It is important to note that the element might potentially be inserted in the document in several different ways, and still meet the constraints defined above. This is analagous to the case when a new file is PUT into a directory on a server; the location of that file within the directory is not specified, and is up to the local file system to decide. The only guarantee is that GET(PUT(x)) returns document x.

If the result of the PUT is a 200 or 202 response, the operation was successful. If it was a 409, the user performed some action which resulted in an invalid document. The 409 response may contain an XML

body, formatted according to the schema in Section 7.2.1.1, which provides further information on the nature of the error. The client MAY use this information to try and alter the request so that this time, it might succeed. The client SHOULD NOT simply retry the request without changing some aspect of it.

6.5 Delete an Element

To delete an element from a document, the client constructs a URI whose document URI points to the document containing the element to be deleted. The node selector MUST be present, and identify the specific element to be deleted.

The client then invokes the HTTP DELETE method. The server will remove the element from the document (including its attributes and its content, such as any children). The client SHOULD be certain, before making the request, that the resulting modified document will also be conformant to the schema.

If the result of the DELETE is a 200 response, the operation was successful. If it was a 409, the user performed some action which resulted in an invalid document. The 409 response may contain an XML body, formatted according to the schema in Section 7.2.1.1, which provides further information on the nature of the error. The client MAY use this information to try and alter the request so that this time, it might succeed. The client SHOULD NOT simply retry the request without changing some aspect of it.

6.6 Fetch an Element

To fetch an element of a document, the client constructs a URI whose document URI points to the document containing the element to be fetched. The node selector MUST be present, and must identify the element to be fetched.

The client then invokes the GET method. The response will contain that XML element. Specifically, it contains the content of the XML document, starting with the opening bracket for the begin tag for that element, and ending with the closing bracket for the end tag for that element. This will, as a result, include all attributes and child elements of that element.

6.7 Create or Replace an Attribute

To create or replace an attribute in an existing element of a document, the client constructs a URI whose document URI points to the document to be modified. The node selector MUST be present. The node selector MUST be constructed such that, if the attribute was

created or replaced as desired, the node selector would select that attribute. If the node selector, when evaluated against the current document, results in a no-match, it is a creation operation. If it matches an existing attribute, it is a replacement operation.

The client then invokes the HTTP PUT method. The content defined by the request MUST be compliant to the grammar for AttValue as defined in XML 1.0. This request MUST be sent with the Content-Type of "application/xml-attribute-value" as defined in Section 10.3. The server will add that attribute such that, if the node selector is evaluated on the resulting document, it returns the attribute present in the request. The client SHOULD be certain, before making the request, that the resulting modified document will also be conformant to the schema.

If the result of the PUT is a 200 or 202 response, the operation was successful. If it was a 409, the user performed some action which resulted in an invalid document. The 409 response may contain an XML body, formatted according to the schema in Section 7.2.1.1, which provides further information on the nature of the error. The client MAY use this information to try and alter the request so that this time, it might succeed. The client SHOULD NOT simply retry the request without changing some aspect of it.

6.8 Delete an Attribute

To delete attributes from the document, the client constructs a URI whose document URI points to the document containing the attributes to be deleted. The node selector MUST be present, and evaluate to an attribute in the document to be deleted.

The client then invokes the HTTP DELETE method. The server will remove the attribute from the document. The client SHOULD be certain, before making the request, that the resulting modified document will also be conformant to the schema.

If the result of the DELETE is a 200 response, the operation was successful. If it was a 409, the user performed some action which resulted in an invalid document. The 409 response may contain an XML body, formatted according to the schema in Section 7.2.1.1, which provides further information on the nature of the error. The client MAY use this information to try and alter the request so that this time, it might succeed. The client SHOULD NOT simply retry the request without changing some aspect of it.

6.9 Fetch an Attribute

To fetch an attribute of a document, the client constructs a URI

whose Document-URI points to the document containing the attribute to be fetched. The node selector MUST be present, containing an expression identifying the attribute whose value is to be fetched.

The client then invokes the GET method. The response will contain an "application/xml-attribute-value" document with the specified attribute, formatted according to the grammar of AttValue as defined in the XML 1.0 specifications.

6.10 Read/Modify/Write Transactions

It is anticipated that a common operation will be to read the current version of a document or element, modify it on the client, and then write the change back to the server. In order for the results to be consistent with the client's expectations, the operation must be atomic.

To accomplish this, the client makes use of entity tags returned by the server in a GET operation used to read the element, attribute, or document that is to be modified. To guarantee atomicity, the PUT operation used to write the changes back to the server MUST contain an If-Match header field, whose value is equal to the entity tag from the prior GET response. If the request fails with a 412 response, the client knows that another update of the data has occurred before it was able to write the results back. The client can then fetch the most recent version, and attempt its modification again.

Because there are no batching operations defined in HTTP that would allow for a number of separate create, modify or delete operations to be performed atomically, designers of application usages should take care to structure their schemas so that operations that need to be performed atomically can be done in a single operation.

6.11 Reading Server Assigned Data

In some application usages, components of the document cannot be set by the user. Rather, they must be filled in by the server. Such cases are documented as part of the application usage. Frequently, the client will want to know the value assigned by the server. As an example, in the resource list application usage [22], the server assigns the uri for a resource list. The client will need this URI to subscribe to the resource list, for example.

There are two ways such discovery can be accomplished. In the first, once the client PUTs a document or element that requires the data to be filled in, the client can do a subsequent GET to find the URI. This GET can be for the entire document, the same URI that was used in the PUT, or a URI that points just to the specific data assigned

by the server. The result of the GET will tell the client about the assigned data. Note that the Etag present in the response is significant, as it will be different from the one returned in the previous response to PUT. That's because, as a result of the server's assignment, the document has changed, and is therefore assigned a new Etag.

The second way a client can learn about the change is through an event package that might be used to find out about changes to XCAP resources.

It is important to note that the 200 OK response to a PUT is always empty, and will not contain the document or element after the server has computed the necessary data.

7. Server Behavior

An XCAP server is an HTTP 1.1 compliant origin server. The behaviors mandated by this specification relate to the way in which the HTTP URI is interpreted and the content is constructed.

An XCAP server MUST be explicitly aware of the application usage against which requests are being made. That is, the server must be explicitly configured to handle URIs for each specific application usage, and must be aware of the constraints imposed by that application usage.

When the server receives a request, the treatment depends on the URI. If the URI refers to an application usage not understood by the server, the server MUST reject the request with a 404 (Not Found) response. If the URI refers to a user that is not recognized by the server, it MUST reject the request with a 404 (Not Found).

Next, the server authenticates the request. All XCAP servers MUST implement HTTP Digest [10]. Furthermore, servers MUST implement HTTP over TLS, RFC 2818 [13]. It is RECOMMENDED that administrators use an HTTPS URI as the XCAP root services URI, so that the digest client authentication occurs over TLS.

Next, the server determines if the client has authorization to perform the requested operation on the resource. The default authorization policy is that only client X can access (create, read, write, modify or delete) resources under the "users/X" directory. Only priviledged administrators can write resources under the "global" directory, but all users can read them.

An application usage can specify an alternate default authorization policy specific to that usage. The server may also know of an application usage that itself defines authorization policies for another application usage. Of course, an administrator or privileged user can override the default authorization policy, although this specification provides no means for doing that.

Once authorized, the specific behavior depends on the method and what the URI refers to.

7.1 POST Handling

Resources managed by XCAP do not represent processing scripts. As a result, POST operations to XCAP URIs are not defined. A server receiving such a request for an xcap resource SHOULD return a 405.

7.2 PUT Handling

The behavior of a server in receipt of a PUT request is as specified in HTTP 1.1 Section 9.6 - the content of the request is placed at the specified location. This section serves to define the notion of "placement" and "specified location" within the context of XCAP resources.

If the request URI represents a document (i.e., there is no node selector component), the content of the request MUST be a valid XML document, and MUST be compliant to the schema associated with the application usage in the URI. If it is not, the request MUST be rejected with a 409 response. If the request URI matches a document that exists on the server, that document is replaced by the content of the request. If the request URI does not match a document that exists on the server, the server adds the document to its repository, and associates it with the URI in the request URI. Note that this may require the creation of one or more "directories" on the server.

If the Request URI represents an XML element (i.e., it contains a node selector, but no attribute selector) the server MUST verify that the document defined by the document URI exists. If no such document exists on the server, the server MUST reject the request with a 404 response code. The content of the request MUST be a single XML element and associated content (including children elements), whose MIME type is "application/xml-fragment-body". If the request does not contain a valid XML fragment body, the request is rejected with a 409 response code. If the request URI matches an element within the document, that element is removed, and replaced with the content of the request. If the request URI does not match an element in the document, the server inserts the content of the request as a new element in the document, such that the resulting document is compliant to the schema, and such that the request URI, when evaluated, would now point to the element which was inserted. There may be more than one way to perform such an insertion; in that case, it is the discretion of the implementor as to how it is done. It may also be possible that the insertion cannot be done because the parent of the element does not exist in the document, or cannot be done because document, after the element is added, would not be compliant to the schema, or because the new element cannot be described by the node-selector no matter what its point of insertion. In such a case, the server MUST return a 409 response code. In all cases, the resulting document MUST be compliant to the schema.

If the Request URI represents an XML attribute (i.e., it contains a node selector and an attribute selector) the server MUST verify that the document defined by the document URI exists. If no such document exists on the server, the server MUST reject the request with a 404

response code. The content of the request will be a MIME object of type "application/xml-attribute-value", which represents a single XML attribute. This attribute will be compliant to the grammar for AttValue as defined in XML 1.0. If the content is not a valid xml-attribute-value, the server rejects the request with a 409 response. If the request URI matches an existing attribute within the document, that attribute is removed, and replaced with the content of the request. If the request URI does not match an attribute in the document, the server inserts the content of the request as a new attribute in the document, such that the resulting document is compliant to the schema, and such that the request URI, when evaluated, would now point to the attribute which was inserted. There may be more than one way to perform such an insertion; in that case, it is the discretion of the implementor as to how it is done. It may also be possible that the insertion cannot be done because the containing element does not exist, or cannot be done because the result of the change would be a document that is not compliant to the schema. In such a case, the server MUST return a 409 response code.

The server MUST check the resulting document for the presence of the "mandatory-schemas" element, which will always be a child of the root element. If this element is present, the server checks each of the schemas listed. If a schema is listed which the server does not support, the server MUST reject the request with a 409 response.

If the application usage indicates that there is a data dependency, the server checks to see if the information contained in the PUT requires the server to compute some component of the document. If it does, the server MUST perform the computation, and then update the document with the result. Since the document has changed, it represents a new instance of the resource, and the server MUST assign a new etag.

If the application usage indicates that there is a data dependency, and that dependency requires the server to perform some kind of data validation beyond that specified in the XML schema, the server MUST perform the validation. If the document fails the validation, the server MUST reject the request with a 409 response. The server MAY add an error report to the response, indicating the nature of the validation error.

If the creation or insertion was successful, the server returns a 200 OK or 201 Created, as appropriate. This response MUST not contain any content.

7.2.1 Detailed Conflict Reports

In cases where the server returns a 409 error response due to any of

the conditions described above, it MAY include a document in the body of the response which provides further details on the nature of the error. This document is an XML document, formatted according to the schema of Section 7.2.1.1. Its MIME type, registered by this specification, is "application/xcap-error+xml".

The document structure is simple. It contains the root element "xcap-error". The content of this element is a specific error condition. Each error condition is represented by a different element. This allows for different error conditions to provide different data about the nature of the error. All error elements support a "phrase" attribute, which can contain text meant for rendering to a human user.

The "schema-validation-error" element indicates that the document was not compliant to the schema after the requested operation was performed. The "not-xml-frag" element indicates that the request was supposed to contain a valid XML fragment body, but did not. Most likely this is because the XML in the body was malformed or not balanced. The "no-parent" element indicates that an attempt to insert an element failed, because the element into which the insertion was supposed to occur did not exist. This element can contain an optional "ancestor" element, which provides an HTTP URI pointed to the xcap resource that identifies the closest ancestor element that does exist in the document. The "cannot-insert" element provides a generic catch-all for other insertion cases. The "no-xml-att-value" element indicates that the request was supposed to contain a valid XML attribute value, but did not. The "no-element" element indicates that an attempt to insert an attribute was made, but the element in which the attribute was to be inserted does not exist.

The "uri-exists" element supports application usages that define data constraints. In particular, it is expected that many application usages will require the server to compute a URI, or allow the client to provide a URI. In either case, the URI needs to be unique within the domain of the server. If the client provides a URI, and the URI already exists, it would be an error. This element describes that condition. For each URI provided by the client which already exists, an "exists" element is present as the content of "uri-exists". Each "exists" element has a "uri" attribute that contains the URI which exists. The "exists" element, in turn, can optionally contain a list of suggested alternate URIs which do not currently exist on the server.

The "unknown-mand-ns" element indicates that the document contained a "mandatory-ns" element that listed a namespace not understood by the server. The content of the "unknown-mand-ns" element is a list of "ns" elements, each one containing a URI identifying a namespace

listed as mandatory in the document, but was not understood by the server.

As an example, the following document indicates that the user attempted to create a resource list using the URI sip:friends@example.com, but that URI already exists:

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xcap-error xmlns="urn:ietf:params:xml:ns:xcap-error"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <uri-exists>
     <exists uri="sip:friends@example.com">
      <alt-uri>sip:friends2@example.com</alt-uri>
     </exists>
    </uri-exists>
   </xcap-error>
```

7.2.1.1 XML Schema

```
   <?xml version="1.0" encoding="UTF-8"?>
   <xs:schema targetNamespace="urn:ietf:params:xml:ns:xcap-error"
xmlns="urn:ietf:params:xml:ns:xcap-error" xmlns:xs="http://www.w3.org/2001/
XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="xcap-error">
     <xs:annotation>
      <xs:documentation>Indicates the reason for the error.</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:choice>
       <xs:element name="schema-validation-error">
        <xs:annotation>
         <xs:documentation>The resulting document was not compliant to the
schema.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="not-xml-frag">
        <xs:annotation>
         <xs:documentation>The request did not contain a valid xml fragment
body.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="no-parent">
        <xs:annotation>
```

```
         <xs:documentation>The element could not be inserted because its parent
does not exist in the document.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ancestor" type="xs:anyURI" minOccurs="0">
           <xs:annotation>
            <xs:documentation>Contains an HTTP URI that points to the element
which is the closest ancestor that does exist.</xs:documentation>
           </xs:annotation>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="cannot-insert">
        <xs:annotation>
         <xs:documentation>The element could not be inserted.</
xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="not-xml-att-value">
        <xs:annotation>
         <xs:documentation>The request did not contain a valid xml attribute
value.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="no-element">
        <xs:annotation>
         <xs:documentation>The attribute could not be inserted because the
element in which to insert does not exist.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="uri-exists">
        <xs:annotation>
         <xs:documentation>The user tried to set a URI that the server must
constrain to be unique, and this URI exists.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
```

```
         <xs:sequence>
          <xs:element name="exists" maxOccurs="unbounded">
           <xs:annotation>
            <xs:documentation>There is an instance of this element for each URI
in the document which suffered this failure.</xs:documentation>
           </xs:annotation>
           <xs:complexType>
```

```
            <xs:sequence minOccurs="0">
             <xs:element name="alt-uri" type="xs:string" maxOccurs="unbounded">
              <xs:annotation>
               <xs:documentation>An optional set of alternate URIs can be
provided.</xs:documentation>
              </xs:annotation>
             </xs:element>
            </xs:sequence>
            <xs:attribute name="uri" type="xs:anyURI" use="required"/>
           </xs:complexType>
          </xs:element>
         </xs:sequence>
         <xs:attribute name="phrase" type="xs:string" use="optional"/>
        </xs:complexType>
       </xs:element>
       <xs:element name="unknown-mand-ns">
        <xs:annotation>
         <xs:documentation>The document had a mandatory namespace which was not
understood by the server.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
          <xs:element name="ns" type="xs:anyURI" maxOccurs="unbounded">
           <xs:annotation>
            <xs:documentation>Each unknown namespace is listed.</
xs:documentation>
           </xs:annotation>
          </xs:element>
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:choice>
     </xs:complexType>
    </xs:element>
   </xs:schema>
```

7.3 GET Handling

The semantics of GET are as specified in RFC 2616. This section clarifies the specific content to be returned for a particular URI that represents an XCAP resource.

If the request URI contains only a document URI, the server returns the document specified by the URI if it exists, else returns a 404 response. The MIME type of the response SHOULD be the most specific type known for that document (i.e., "application/resource-lists+xml" instead of "application/xml"). If the request URI contains a node selector, and that node selector identifies an XML element in an existing document, that element is returned in the 200 response. The MIME type of the response MUST be "application/xml-fragment-body". The content of the response is the portion of the XML document starting with the left bracket of the begin tag of the element, ending with the right bracket of the end tag of the element. If the request URI contains a node selector, and that node selector contains an attribute selector, and that attribute exists in the specified document, the server returns that attribute. The MIME type of the response MUST be "application/xml-attribute-value", which contains an XML attribute value formatted according to the grammar of AttValue in the XML 1.0 specifications. In all cases, if the referenced resource does not exist, a 404 is returned.

7.4 DELETE Handling

The semantics of DELETE are as specified in RFC 2616. This section clarifies the specific content to be deleted for a particular URI that represents an XCAP resource.

If the request URI contains only a Document-URI, the server deletes the document specified by the URI if it exists and returns a 200 OK response, else returns a 404 response. If the request URI specifies a Node-Selector, the server verifies that the document specified by the Document-URI exists. If it does not exist, the server returns a 404 (Not Found) response. If the document does exist, and the node selector specifies an XML element that exists, that element is removed from the document. If the document does exist, and the node selector specifies an XML attribute that exists in the document, that attribute is removed from the document. If the node selector returns a no-match, a 404 (Not Found) is returned. However, if removal of the element or attribute would result in a document which does not comply with the XML schema for the application usage, the server MUST NOT perform the deletion, and MUST reject the request with a 409 (Conflict).

7.5 Managing Etags

An XCAP server MUST maintain entity tags for all resources that can be referenced by a URI within a particular document. RFC 2616 allows for entity tags for one resource to be applied to other resources. In the case of XCAP resources, a server MUST use the same etag to reference all resources that define elements within a particular document. In other words, the server effectively maintains a single etag per document, and all resources within that document inherit the same etag.

This constraint is necessary for a client to make changes to various parts of a document even though it only posesses the etag for the

overall document.

8. Examples

This section goes through several examples, making use of the resource-lists [22] XCAP application usage.

First, a user Bill creates a new document (see Section 6.1). This document is a new resource-list, initially with no users in it:

```
PUT
http://xcap.example.com/services/presence-lists/users/bill/fr.xml HTTP/1.1
Content-Type:application/presence-lists+xml

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="friends" uri="sip:friends@example.com" subscribeable="true">
  </list>
</resource-lists>
```

Next, Bill creates an element in this document (Section 6.4). In particular, he adds an entry to the list:

```
PUT
http://xcap.example.com/services/presence-lists/users/bill/fr.xml/
resource-lists/list[@name="friends"]/entry HTTP/1.1
Content-Type:application/xml-fragment-body

<entry name="Bob" uri="sip:bob@example.com">
  <display-name>Bob Jones</display-name>
</entry>
```

Next, Bill fetches the document (Section 6.3):

```
GET
http://xcap.example.com/services/presence-lists/users/bill/fr.xml HTTP/1.1
```

And the result is:

```
HTTP/1.1 200 OK
Etag: "wwhha"
Content-Type: application/presence-lists+xml

<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <list name="friends" uri="sip:friends@example.com"
       subscribeable="true">
    <entry name="Bob" uri="sip:bob@example.com">
      <display-name>Bob Jones</display-name>
    </entry>
  </list>
</resource-lists>
```

Next, Bill adds another entry to the list, which is another list that has three entries. This is another element creation (Section 6.4):

```
PUT
http://xcap.example.com/services/presence-lists/users/bill/fr.xml/
presence-lists/list[@name="friends"]/list[@name="close-friends"] HTTP/1.1
Content-Type: application/xml-fragment-body

<list name="close-friends" uri="sip:close-friends@example.com"
      subscribeable="true">
   <entry name="Joe" uri="sip:joe@example.com">
     <display-name>Joe Smith</display-name>
   </entry>
   <entry name="Nancy" uri="sip:nancy@example.com">
     <display-name>Nancy Gross</display-name>
   </entry>
   <entry name="Petri" uri="sip:petri@example.com">
     <display-name>Petri Aukia</display-name>
   </entry>
</list>
```

Then, Bill decides he doesnt want Petri on the list, so he deletes the entry (Section 6.5):

```
DELETE
http://xcap.example.com/services/presence-lists/users/bill/fr.xml/
presence-lists/list/list/entry[@name="Petri"] HTTP/1.1
```

Bill decides to check on the URI for Nancy, so he fetches a particular attribute (Section 6.6):

```
GET
http://xcap.example.com/services/presence-lists/users/bill/fr.xml/
presence-lists/list/list/entry[@name="Nancy"]/@uri HTTP/1.1
```

and the server responds:

```
HTTP/1.1 200 OK
Etag: "ad88"
Content-Type:application/xml-attribute-value

"sip:nancy@example.com"
```

9. Security Considerations

Frequently, the data manipulated by XCAP contains sensitive information. To avoid eavesdroppers from seeing this information, it is RECOMMENDED that an admistrator hand out an https URI as the XCAP root services URI. This will result in TLS-encrypted communications between the client and server, preventing any eavesdropping.

Client and server authentication are also important. A client needs to be sure it is talking to the server it believes it is contacting. Otherwise, it may be given false information, which can lead to denial of service attacks against a client. To prevent this, a client SHOULD attempt to upgrade [14] any connections to TLS. Similarly, authorization of read and write operations against the data is important, and this requires client authentication. As a result, a server SHOULD challenge a client using HTTP Digest [10] to establish its identity, and this SHOULD be done over a TLS connection.

10. IANA Considerations

There are several IANA considerations associated with this specification.

10.1 XCAP Application Usage IDs

This specification instructs IANA to create a new registry for XCAP application usage IDs (AUIDs).

XCAP AUIDs are registered by the IANA when they are published in standards track RFCs. The IANA Considerations section of the RFC must include the following information, which appears in the IANA registry along with the RFC number of the publication.

Name of the AUID. The name MAY be of any length, but SHOULD be no more than twenty characters long. The name MUST consist of alphanum [15] characters only.

Descriptive text that describes the application usage.

10.2 application/xml-fragment-body MIME Type

This specification registers a new MIME type according to the procedures of RFC 2048 [7] and guidelines in RFC 3023 [8].

MIME media type name: application

MIME subtype name: xml-fragment-body

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [8].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [8].

Security considerations: See Section 10 of RFC 3023 [8].

Interoperability considerations: none.

Published specification: The XML Fragment Interchange [4], which defines an XML fragment body as a well-balanced region of an XML document being considered as (logically and/or physically) separate from the rest of the document for the purposes of defining it as a fragment.

Applications which use this media type: This document type has been used to support transport of XML fragment bodies in RFC XXXX [[NOTE TO RFC EDITOR: Please replace XXXX with the published RFC number of this specification.]], the XML Configuration Access Protocol (XCAP).

Additional Information:

Magic Number: None

File Extension: .xfb or .xml

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, jdrosen@jdrosen.net

Intended usage: COMMON

Author/Change controller: The IETF.

10.3 application/xml-attribute-value MIME Type

This specification registers a new MIME type according to the procedures of RFC 2048 [7] and guidelines in RFC 3023 [8].

MIME media type name: application

MIME subtype name: xml-attribute-value

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [8].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [8].

Security considerations: See Section 10 of RFC 3023 [8].

Interoperability considerations: none.

Published specification: An entity of this MIME type is compliant to the grammar for AttValue as specified in XML 1.0 [1].

Applications which use this media type: This document type has been used to support transport of XML attribute values in RFC XXXX [[NOTE TO RFC EDITOR: Please replace XXXX with the published RFC number of this specification.]], the XML Configuration Access Protocol (XCAP).

Additional Information:

Magic Number: None

File Extension: .xav

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, jdrosen@jdrosen.net

Intended usage: COMMON

Author/Change controller: The IETF.

10.4 application/xcap-error+xml MIME Type

This specification registers a new MIME type according to the procedures of RFC 2048 [7] and guidelines in RFC 3023 [8].

MIME media type name: application

MIME subtype name: xcap-error+xml

Mandatory parameters: none

Optional parameters: Same as charset parameter application/xml as specified in RFC 3023 [8].

Encoding considerations: Same as encoding considerations of application/xml as specified in RFC 3023 [8].

Security considerations: See Section 10 of RFC 3023 [8].

Interoperability considerations: none.

Published specification: This specification.

Applications which use this media type: This document type conveys error conditions defined in RFC XXXX. [[NOTE TO RFC EDITOR: Please replace XXXX with the published RFC number of this

specification.]]

Additional Information:

Magic Number: None

File Extension: .xe

Macintosh file type code: "TEXT"

Personal and email address for further information: Jonathan Rosenberg, jdrosen@jdrosen.net

Intended usage: COMMON

Author/Change controller: The IETF.

10.5 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:xcap-must-understand

This section registers a new XML namespace, as per the guidelines in RFC 3688 [16].

URI: The URI for this namespace is urn:ietf:params:xml:ns:xcap-must-understand

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Resource Lists Namespace</title>
</head>
<body>
  <h1>Namespace for XCAP Must Understand Element</h1>
  <h2>urn:ietf:params:xml:ns:xcap-must-understand</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
```

```
</html>
END
```

10.6 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:xcap-error

This section registers a new XML namespace, as per the guidelines in RFC 3688 [16].

URI: The URI for this namespace is urn:ietf:params:xml:ns:xcap-error

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
          "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Resource Lists Namespace</title>
</head>
<body>
  <h1>Namespace for XCAP Error Documents</h1>
  <h2>urn:ietf:params:xml:ns:xcap-error</h2>
  <p>See <a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
END
```

10.7 XCAP Error Schema Registration

This section registers an XML schema per the procedures in [16].

URI: please assign.

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

The XML for this schema can be found as the sole content of Section 7.2.1.1.

10.8 XCAP Mandatory Namespace Schema Registration

This section registers an XML schema per the procedures in [16].

URI: please assign.

Registrant Contact: IETF, SIMPLE working group, (simple@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

The XML for this schema can be found as the sole content of Section 4.7.1.

11. Acknowledgements

The author would like to thank Ben Campbell, Eva-Maria Leppanen, Hisham Khartabil, and Chris Newman for their input and comments.

Normative References

[1] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler, "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C FirstEdition REC-xml-20001006, October 2000.

[2] Thompson, H., Beech, D., Maloney, M. and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC REC-xmlschema-1-20010502, May 2001.

[3] Bray, T., Hollander, D. and A. Layman, "Namespaces in XML", W3C REC REC-xml-names-19990114, January 1999.

[4] Grosso, P. and D. Veillard, "XML Fragment Interchange", W3C CR CR-xml-fragment-20010212, February 2001.

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[6] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[7] Freed, N., Klensin, J. and J. Postel, "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", BCP 13, RFC 2048, November 1996.

[8] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[9] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", W3C REC REC-xpath-19991116, November 1999.

[10] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[11] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[12] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[13] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[14] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[15] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[16] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

Informative References

[17] Rosenberg, J., "A Presence Event Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-presence-10 (work in progress), January 2003.

[18] Rosenberg, J., "A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP)", draft-ietf-simple-winfo-package-05 (work in progress), January 2003.

[19] Rosenberg, J., "An Extensible Markup Language (XML) Based Format for Watcher Information", draft-ietf-simple-winfo-format-04 (work in progress), January 2003.

[20] Roach, A., Rosenberg, J. and B. Campbell, "A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists", draft-ietf-simple-event-list-04 (work in progress), June 2003.

[21] Rosenberg, J. and M. Isomaki, "Requirements for Manipulation of Data Elements in Session Initiation Protocol (SIP) for Instant Messaging and Presence Leveraging Extensions (SIMPLE) Systems", draft-ietf-simple-data-req-03 (work in progress), June 2003.

[22] Rosenberg, J., "An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Usage for Presence Lists", draft-ietf-simple-xcap-list-usage-01 (work in progress), October 2003.

[23] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[24] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[25] Roach, A., "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265, June 2002.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.